Amendments to the Specification

Please amend the following paragraphs as specified.

Please amend paragraph [0032] as follows:

The example routine is written in a C language format pseudocode. This code is for illustrating the basic operation of an exception handling routine and cannot be compiled.

```
/* The hardware exception logic transfers program execution to a preprogrammed address where a software exception handler is stored. For this example the exception handler is called Exception_handler. The bad address is data that caused a hardware exception condition in the core 100.
*/
Exception_handler()
{
    /* Check to see if this is an exception for a bad address.
    */
    if (conduct hardware dependent check for bad address);
    {
        /*
        The hardware exception logic passes the bad address to the software exception handler (the detailed method is hardware dependent). The software exception handler uses the bad address to determine if the exception is a decode trigger or an actual bad address. To do this the hardware may have a list of valid
```

addresses that are acceptable or a range of addresses that are acceptable or just check to see if it is an odd address (lowest order address bit set).

*/

if (*valid decode address*);

{

/*

If the software exception handler determines the cause of the exception is a purposeful decode trigger, it may use the bad address to determine where the encoded data is stored and decode that encoded data into a location in memory from which it can later be executed. If a transformation of the data is desired, a function, decode, is ~~call~~ called to perform that function. The decode function will decide when to stop decoding. For example if this were a program instruction, being decoded, a good stopping point would be the next branch instruction. The decode function will return a value, decode_return. This value could represent an address of a decoded section of code that the core 100 can use to continue execution after it leaves the decode function or as an address where the newly decoded data is stored.

*/

decode_return = decode(bad_address)

/*

What the software exception handler does depends on what was decoded. For this example, the decoder decoded compressed

instructions and the decode_return value was the address of the first instruction of a block of instructions that were decoded. The decode function will need to "fix" the address so the core 100 will know where to begin execution after the exception handling function finishes. In many core 100 designs there is a register in the core 100 called a program ~~counter112~~ counter 112 which is the address the core 100 uses to fetch the current instruction or the next instruction that will be executed. The program counter 112 may still contain the bad address that caused the exception. This address needs to be changed to the address of the decompressed code so the core 100 will begin executing [[~~of~~]] the decompressed code.

*/

fix_program_counter(decode_return);

}

else

{

/*

If this was not a valid decode address or exception then the exception function will continue here with logic to process the exception.

*/

}

}

else

```
{
        /*
        Check for other exceptions
        */
}
/*
At this point the exception function will do any additional housekeeping that is
needed for the particular core 100 architecture and return.
*/
return;
}
```

Please amend paragraph [0038] as follows:

In some embodiments in which the invention is implemented using software, the software may be stored as computer program product and loaded into computer system 600 using removable storage drive ~~614~~ 613, hard drive 612 or communications interface 624. The control logic (software), when executed by the processor 605, causes the processor 605 to perform the functions of the invention as described herein.

Please amend the Abstract as follows (a clean copy of which is attached hereto as a separate sheet):

~~Processor instructions are compressed and stored in computer system memory.~~ When ~~the~~ processor ~~instruction~~ instructions are required for ~~execution~~ execution, a misaligned address is sent to the processor. The misaligned instruction address causes a computer processor exception. The computer system automatically executes an exception handling routine that transforms ~~the compressed instructions~~ data into at least one executable instruction[[s]] for the processor. In embodiments, data is transformed by decompressing a compressed instruction, decrypting an encrypted instruction, decoding a macro instruction, or transforming a non-native instruction into at least one instruction.